STA602_HW1

Yutong Shao

2023-01-15

1. Working with data

a. load data set

```
# use read.table to read .dat file
rain.df <- read.table('rnf6080.dat')
```

b. number of rows and columns

`rain.df` has 5070 rows and 27 columns.

```
# There two methods
# 1. report the dimension of the dataframe
dim(rain.df)
```

```
## [1] 5070   27
```

```
# 2. report the number of columns and rows separately
ncols <- ncol(rain.df)
nrows <- nrow(rain.df)
print(ncols)
```

```
## [1] 27
```

```
print(nrows)
```

```
## [1] 5070
```

c. names of columns

```
# use colnames() function to report column names
print(colnames(rain.df))
```

```
##  [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11" "V12"
## [13] "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22" "V23" "V24"
## [25] "V25" "V26" "V27"
```

d.

To get the value of certain index, simply use [ ] with the first number being row index and the second number being column index. The value at row2, column 4 is 0.

```
# get the value at the second row, fourth column
rain.df[2,4]
```

```
## [1] 0
```

e. display the whole second row

```
# show the entire second row
# the space after comma means returning the entire columns
print(rain.df[2,])
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21
## 2 60  4  2  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0
##   V22 V23 V24 V25 V26 V27
## 2   0   0   0   0   0   0
```

f.

```
# original column names
print('Original:')
```

```
## [1] "Original:"
```

```
names(rain.df)
```

```
##  [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11" "V12"
## [13] "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22" "V23" "V24"
## [25] "V25" "V26" "V27"
```

```
# reset column names
names(rain.df) <- c("year","month","day",seq(0,23))
newcolnames <- colnames(rain.df)

# verify new column names
print('New:')
```

```
## [1] "New:"
```

```
print(colnames(rain.df))
```

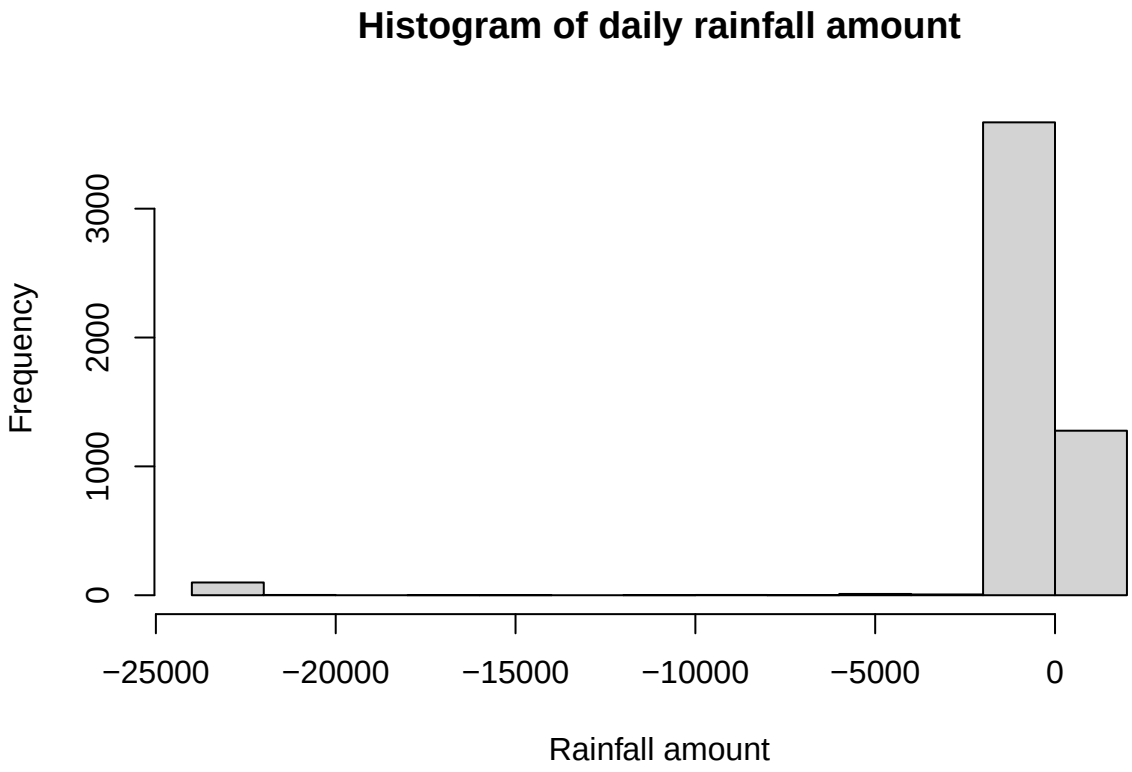
```
##  [1] "year"  "month" "day"   "0"     "1"     "2"     "3"     "4"     "5"
## [10] "6"     "7"     "8"     "9"     "10"    "11"    "12"    "13"    "14"
## [19] "15"    "16"    "17"    "18"    "19"    "20"    "21"    "22"    "23"
```

This command is renaming the columns, with the first three being “year”,“month”,“day”, and the rest are consecutive numbers from 0 to 23.

g. create a new column

```r
# creating a new column named "daily" which equals the sum of
# column "0" to "23"
rain.df$daily <- rowSums(rain.df[, c(4:ncols)])
```

h. histogram of daily rainfall amounts



i.

There are negative amounts of rainfall in the histogram, which is unrealistic. This is because there are one or more missing values that has been marked as -999 on certain days. So, the histogram above cannot possibly be right.

j. Fix the data frame

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

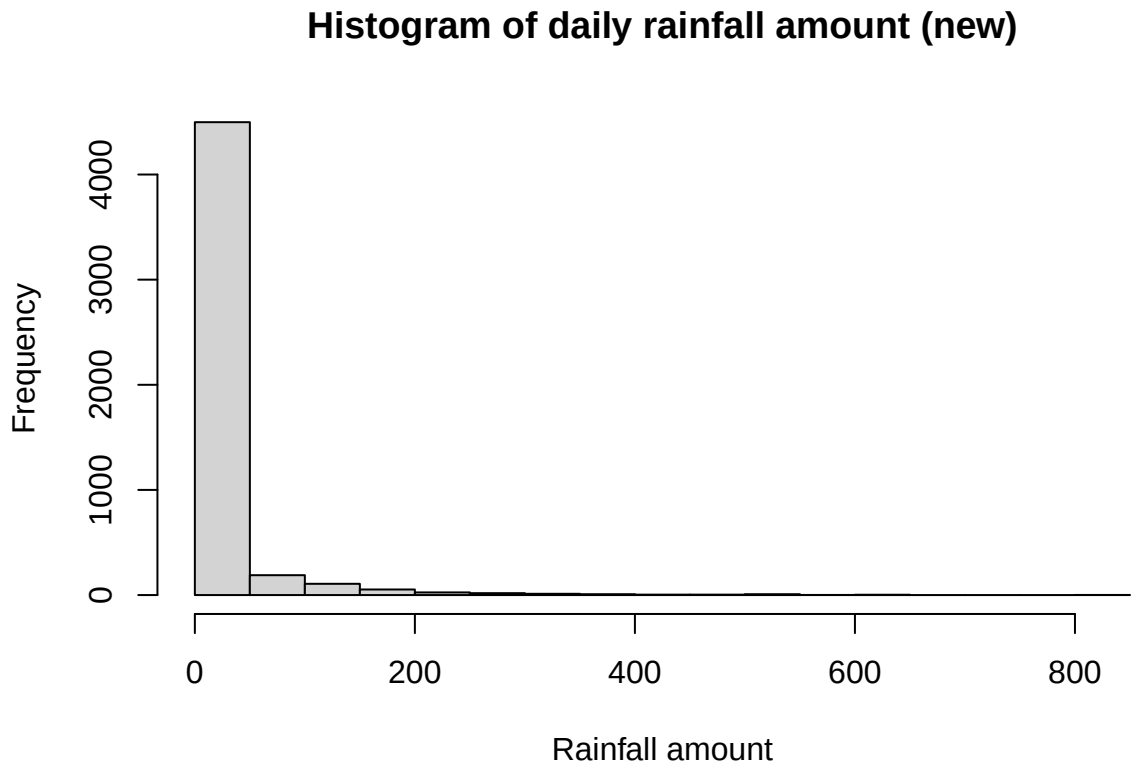
```r
# remove missing values (here we remove all rows
# that contain value of -999)
newdf <- filter_all(rain.df, all_vars(.!= -999))
print('number of obs after removing missing values:')
```

```
## [1] "number of obs after removing missing values:"
```

```r
nrow(newdf)
```

```
## [1] 4931
```

```r
# calculate daily rainfall again
newdf$new_daily <- rowSums(newdf[, c(4:ncols)])
hist(newdf$new_daily,
     main='Histogram of daily rainfall amount (new)',
     xlab='Rainfall amount')
```



The new histogram is more reasonable because there is no negative values, and during most days, the rainfall amount is around 0-50, which is close to reality.

2. Data types

a.

```r
x <- c("5", "12", "7")
# this line creates a vector with three characters

max(x)
```

```
## [1] "7"
```

```r
# This line finds the 'max' of three characters,
# The sorting rule is comparing each character in each string one by one,
# for example, "7" is the largrest.
# If encounter ties, compare the immediate next character
# following the same rule

sort(x)
```

```
## [1] "12" "5"  "7"
```

```r
# this line sorts the characters following the above rule

# sum(x)    # Data type error: cannot add up characters
```

b.

```r
y <- c("5",7,12)
# The output is a vector of three characters
# this is because if the inputs of c() are of different data types,
# they will all be transformed to the type of the first element,
# which is character here.

print(y)
```

```
## [1] "5"  "7"  "12"
```

```r
print(class(y[2]))
```

```
## [1] "character"
```

```r
print(class(y[3]))      # verify the data type
```

```
## [1] "character"
```

```r
# y[2] and y[3] are all characters

# y[2] + y[3]
# Data type error: cannot add up characters
```

c.

```r
z <- data.frame(z1="5",z2=7,z3=12)
# This line is creating a dataframe with the first column being a character, "5",
# the second being a number 7, and the third being a number 12,
# column names are z1, z2, z3.

z[1,2] + z[1,3]
```

```
## [1] 19
```

```r
# this line is adding up the second and third value in the first row,
# i.e. 7 + 12 = 19
```

3.

a.) A reproducible code is the code that can run directly by copying and pasting.

b.) Reproducible code will be convenient for TAs to verify whether my code make sense and also ensure me getting full scores. For example, imagine I am discussing with my classmates on the homework, if my code cannot be rerun successfully, we will spend extra time debugging, which is inefficient.

c.) 2, this assignment helps us review basic R programming.